16 Summary and conclusions

Congratulations, you made it all the way through this book! We hope you have enjoyed reading it as much as we did writing it. We covered quite a lot of material on our trip, as OAuth isn't a simplistic protocol and there are many moving pieces and a variety of ways to put them together. This seems daunting at first, but we hope that now that you're armed with the knowledge of how everything works, you can see that it's as simple as possible but no simpler. We hope that what you've learned while building all the moving pieces of OAuth with us will guide you in your future expeditions. In this final chapter, we're going to take a few pages to talk about the bigger picture.

16.1 The right tool

OAuth is a powerful delegation protocol, but we know that you're not using OAuth for its own sake. The "auth" in OAuth stands for "authorization," and nobody uses an authorization protocol unless they're trying to authorize some kind of action. We know that you likely first had to use OAuth to do *something else*: something useful, something beautiful, something wonderful. OAuth is merely one of many tools at your disposal for solving a certain set of problems, whether protecting an API, building a client to access an API, or developing an overarching security architecture to tie a system together. As with all tools, it's useful and important to understand how

[redacted]

those tools work and what they're good for. After all, although it's possible to drive a screw into a wall using a hammer, you'll have much better luck with a screwdriver. We want you to be able to know when OAuth is a good fit for your problem and, just as important, when it's not a good fit.

It's our hope that this book has given you the kind of in-depth knowledge you'll need to recognize when to reach for OAuth in the toolbox and how to apply it to your particular set of problems. We didn't want this book to be merely a guide for the OAuth implementation at the leading internet providers, or even how to use a particular OAuth library. We believe there already exists plenty of documentation online for those kinds of focused queries. Instead, we wanted you to be so thoroughly familiar with OAuth by the end that you'll be able to use it no matter what platform or application you're using it on.

We know that almost no one reading this book will implement an OAuth ecosystem from scratch, end to end. So why did we go through all the trouble? The text and exercises in this book should leave you with a deep understanding and appreciation for the OAuth protocol and all of its attendant pieces, allowing you to see the data as it flows through the system. By building out an entire ecosystem from scratch, we hope you'll have a better idea of what's going on when your client sends out a request and gets back a peculiar response, or why clients are sending things to your resource server in ways that you didn't quite expect. By working on the implementation from the other side, we hope you'll be more tolerant and understanding of how things are built, but not *so* tolerant that you open up an inadvertent security hole. After all, by now it should be clear that the parts of the OAuth process are put there for good reason.

If instead of reading this entire book cover to cover you jumped into only the part that seemed most relevant to your immediate problem, we won't judge you. In fact, that's how we tend to treat big technical books like this one. However, now would be a great time to go back and look at other parts of the system. Are you building a client? Then go try to build out the authorization server. Are you working on an in-browser application? Check out the native application section. Are you working on an authentication protocol like OpenID Connect? Go back and check out the guts of the OAuth protocol itself and get a feel for how that authentication information is carried across the network in OIDC.

16.2 Making key decisions

As we've seen throughout the book, OAuth is a framework that's full of options. Navigating these options is a little tricky, but we hope that by now you've got a good idea of what to do in this space. It's our hope that you can also see the value in each of the different options provided by the OAuth framework and why those options were added. Instead of having a single protocol that doesn't quite fit well to any of its major uses, OAuth 2.0 gives us a bunch of pieces that we can use to do different things. We hope this book provides you with a means of deciding which pieces to apply and when to apply them.

[redacted]

We realize that it's tempting to take shortcuts in this process, going with something that feels simpler instead of the options that are the best fit for your use case. After all, the implicit flow is simple, why not use that for everything and skip the authorization code nonsense? Or why bother the user at all when we can add a parameter to our API and have the client tell us which user it's acting on behalf of? However, as we saw in the vulnerabilities section of this book, these kinds of shortcuts inevitably lead to pain and compromises. As enticing as it may seem, there really are no good shortcuts to good security.

One of the most important decisions to make is whether to use OAuth 2.0 at all. The flexibility and ease of implementation that make OAuth the go-to protocol for protecting APIs also make it tempting to treat OAuth as the *appropriate* solution to many different problems. The good news here is that the structure and philosophy of OAuth can be ported, as has been shown with recent efforts to deploy the process on non-HTTP protocols. Corollary to this is whether OAuth is a *sufficient* solution for a problem: as we've seen, many times OAuth in the real world is augmented with other technologies to solve a larger problem than simple rights delegation. We contend that this is a good thing, and leads to a rich ecosystem.

Once you've decided to build out an OAuth system, there's one question that you should be asking first: Which authorization grant type should you be using? The authorization code grant type that we went into great detail in chapter 2 and built over the course of chapters 3 through 5 should be the default choice for most situations. Only if what you're building fits into one of several specific optimizations should you be considering a different grant type. A large part of that optimization space depends on what kind of client you're building (or expecting to be built) to access an API. For instance, if your client is running entirely within a browser, then the implicit flow makes sense. But if you're building a native application, the authorization code flow remains much better. If you're not acting on behalf of a particular user, then the client credentials flow will work well for you. But if you are acting on behalf of a specific user, then you're much better off involving that user in the process using one of the interactive grant types. This is true even if the user doesn't make the authorization decision, as in an enterprise deployment, because the presence of an authenticated user can be used as one major pillar in the overall security architecture.

Of course, even if you make every core decision correctly, things can still go wrong in deployment or use. That's the nature of security, and it doesn't come for free. OAuth goes a long way in helping people get things right by pushing the complexity away from the clients and on to the authorization servers, but even then it's important to use these components in the right way. This includes making sure all of the surrounding and underlying systems, such as TLS, are up and running as intended. OAuth helps here, too, by basing its security model on many years of deployment experience across a wide variety of systems. By following a few best practices and using the right parts of OAuth in the right way, a small one-off API provider can give access

[redacted]

to the same kind of security and delegation capabilities available on the largest APIs of the internet today.

16.3 The wider ecosystem

As we've shown throughout this book, OAuth was designed to do a job—security delegation—and do that job well. OAuth doesn't handle some things well, or at all, but as we've seen in the latter parts of this book, that's not a bad thing. OAuth has provided a solid base upon which a wide and rich ecosystem has been erected over time. Much in the same way that the OAuth framework provides a set of choices to solve different problems, the surrounding ecosystem provides a set of choices of things that can be used along with OAuth to solve a wide variety of requirements.

The OAuth working group explicitly left out several items key to a real security architecture, such as the format of the access token itself. After all, if you need to create the tokens anyway, why not tell everybody exactly how to make the tokens? This conscious omission has allowed for the development of complementary technologies such as JOSE and JWT that work well with, but aren't dependent upon, OAuth. JOSE combines the simplicity of JSON with advanced cryptographic functionality in a way that's still easy for developers to get right. Alternatively, for situations in which the tokens need to be compact, or don't need to be self-contained at all, token introspection provides a viable alternative. Importantly, these two can be swapped out or even combined without the client being any the wiser, all thanks to OAuth's commitment to do one job and one job alone.

Other extensions are added to OAuth to deal with specific situations. For instance, the PKCE extension we described in chapter 10 is used to help prevent the theft of the authorization code on native mobile applications using a local application-specific URI scheme. PKCE doesn't make much sense outside of this narrow window, but it works wonderfully within that space. Likewise, the token revocation extension provides a way for clients to proactively throw away tokens. Why is this not a universal feature of OAuth? In some systems, the OAuth token is entirely self-contained and stateless, and there isn't a reasonable way to propagate a "revocation" event through the distributed system of resource servers. In others, the clients are presumed to be completely naïve to the status of the token, and this kind of proactive measure would make little sense. However, for ecosystems in which the tokens are stateful and the clients are marginally smarter than they need to be, revocation provides an extra bit of security on top of everything else.

Comparably, OAuth can be used to build lots of different protocols. We saw in great detail in chapter 13 that although OAuth is not itself an authentication protocol, it can be used to build one. Furthermore, we saw how profiles and applications of OAuth such as HEART and iGov can bring together large communities for interoperability. These efforts lock down a set of OAuth's choices and extensions in a way that provides a pattern and recipe that others can follow. OAuth can also be used to build complex

[redacted]

protocols such as UMA that take the OAuth delegation pattern and extend it to multiple parties.

16.4 The community

OAuth has a thriving community on the internet, with lots of resources, discussions, and experts ready to help out. As you've seen in this book, there are a lot of decisions to be made when building an OAuth ecosystem, and chances are there's somebody else on the internet that's already made that decision in similar circumstances. You'll find open source projects, large companies, nimble startups, waves of consultants, and (we hope) at least one decent book that will help you figure out what OAuth is all about. You're far from alone in your choice of this security system, and when it comes to security, there is strength in many eyes and many hands on the problem space.

The OAuth working group is still working on the protocol and its extensions, and you (yes, you) can join the conversation.[1] Think you can do better than OAuth? Does your use case need some bit of special sauce on top of OAuth? Do you need a grant type with different assumptions and deployment characteristics? Then by all means bring the conversation to the working group.

Since OAuth isn't defined and owned by a company, or even a consortium like many open source projects, it doesn't have official branding and marketing to make it recognizable. Instead, the community has once again stepped in. OAuth's bus token logo was drawn by Chris Messina and submitted to the community, where it took off even though it's technically not official. The OAuth protocol even has an unofficial-yet-awesome mascot: OAuth-tan (figure 16.1)![2]

This community has been able to grow naturally because of a few things. First and foremost, OAuth solves a real problem that people have. The API and mobile application economies were only getting started when OAuth 1.0 and 2.0 were developed, and OAuth stepped in to provide a much needed security layer. Second, and just as important, OAuth is an open protocol. It's not owned or controlled by any one company, and anybody can build an implementation without paying royalties or licensing fees. This means that there's grown up a large community of software, libraries, and demonstrations that support OAuth out of the box in its various forms. What if there isn't something on your favorite platform? Chances are that there's a similar implementation that you can copy from another language, all ready to go.

OAuth is relatively simple. Sure, you've gone through somewhat north of 300 pages of material on the topic, and there are a lot of things that can go quickly wrong with it if you're not careful. But compared with the deep complexities of many of the systems that have come before (Kerberos, SAML, or WS-*), it's downright trivial. This is especially true, we've seen, of clients, which make up the majority of the ecosystem. The balance of simplicity has allowed it to be adopted in ways and in places that have previously eschewed security protocols, bringing more developers into the fold.

[1] https://tools.ietf.org/wg/oauth/
[2] Used by permission from Yuki Goto.

[redacted]

Figure 16.1 OAuth-tan! We *dare* you to access the APIs she's protecting without her permission.

16.5 The future

The OAuth Dance that we saw in chapter 2 is one of the most popular dances on the internet today, but today's popularity is in no way guaranteed going forward. Although we do think that the power and relative simplicity of OAuth 2.0 mean that it will be around for a while yet, technology inevitably marches onward.

In the OAuth world itself, we've already seen some of this with the effort in standardizing PoP tokens. The PoP specifications have taken the JOSE technologies and applied them in a new way to OAuth, creating PoP tokens that will provide a higher level of security than bearer tokens, but at the cost of some added complexity. As such, they'll likely be deployed as needed, and often alongside of bearer tokens. It's possible we'll see other forms of tokens in the future, such as ones bound to the TLS layer or using exotic new cryptographic mechanisms.

Enough extensions and components have been added to OAuth that it's likely that we'll see an OAuth 2.1 or OAuth 3.0 that combines all of these components into a single, cohesive suite of specifications that are easier to navigate than the organically grown documents we have today. Any such reorganization work is a long way in the future, if it's to happen at all.

Finally, there will almost inevitably come a day when OAuth is itself entirely supplanted by something newer and better. OAuth is a child of its time, with direct dependencies on HTTP, web browsers, and JSON. Although their life spans may be long, these technologies won't continue to exist in the same universality and form as they do today. In much the same way that OAuth has replaced Kerberos, WS-Trust, and SAML in many security architectures and fit into new places where those older protocols could not, there will be something new that will eventually take OAuth's place as the security protocol to pay attention to.

Until then, OAuth is and will continue to be well worth understanding and building.

16.6 Summary

It was quite a journey. We started with the core definition of OAuth 2.0, wound our way up through the actors and components and interconnections, and built an entire ecosystem from scratch. Then we took a step back and looked at what could go wrong, and

[redacted]

how to fix it. We then took a more in-depth look at the world of protocols surrounding OAuth, including OpenID Connect and UMA. Finally, we looked ahead to where OAuth might be going in the future with PoP tokens and token binding.

What's next? Now, it's time to build *your* system. Find good libraries. Contribute to open source. Engage in the standards community. After all, you're not building OAuth capability for its own sake; you're building OAuth capability that you'll use to protect and secure some other piece of functionality that you and the rest of the world care about. Now that you've got a handle on how delegation, authorization, and many of the attendant security bits work using this OAuth thing, you can concentrate on what you're really after: building your application, API, or ecosystem.

Thank you for joining us. We hope you have enjoyed the trip as much as we've enjoyed guiding it.

[redacted]

appendix A
An introduction to our code framework

A.1 An Introduction to Our Code Framework

Throughout this book, we'll be developing applications in JavaScript using the Express.js[1] web application framework running on Node.js,[2] a server-side JavaScript engine. Although the examples themselves will be written in JavaScript, all concepts in the examples should be readily portable to other platforms and application frameworks. Wherever possible, we've tried to keep the specific quirks of JavaScript (such as closures and function callbacks) away from the bits of code that you'll be working with directly, since the goal of this book is not to make you a proficient JavaScript programmer. We also will be making use of library code for non-OAuth-specific functionality in these examples so that you can focus on what is the core goal of this book: understanding in detail how the OAuth protocol works.

In a real application, it may be desirable to use an OAuth library to handle many of the functions that we'll be coding by hand here. However, in this book, we'll be building things up by hand so that you can get hands on with the OAuth functionality without getting sidetracked by the specifics of the Node.js application. All of the

[1] http://expressjs.com/
[2] https://nodejs.org/

[redacted]

code found in this book is available on GitHub[3] and from the publisher's site for this book.[4] Each exercise is in a separate directory, sorted by chapter number and example number.

Let's get started. First, before you'll be able to run anything, you'll need to install Node.js and the Node Package Manager (NPM) for your platform. The details vary from system to system; for example, on a MacOSX system running MacPorts, this can be installed with the following commands:

```
> sudo port install node
> sudo port install npm
```

You can verify that these have been properly installed by asking each for their version number, which will print out something like the following messages:

```
> node -v
v4.4.1
> npm -v
2.15.1
```

With these core libraries installed, we can unpack our example code. Enter the `ap-A-ex-0` directory and run the `npm install` command to install the project dependencies for this example. This action pulls down the dependencies for this example and installs them into the `node_modules` directory. The npm program will detail all of the packages being installed automatically to satisfy the dependencies for this project, and the output will look something like this:

```
ap-A-ex-0> npm install
underscore@1.8.3 node_modules/underscore

body-parser@1.13.2 node_modules/body-parser
├┼┼ content-type@1.0.1
├┼┼ bytes@2.1.0
```

There's a lot of information that will print to your console here; we're not going to copy everything.

```
├┼┼ send@0.13.0 (destroy@1.0.3, statuses@1.2.1, ms@0.7.1, mime@1.3.4, http-
   errors@1.3.1)
├┼┼ accepts@1.2.11 (negotiator@0.5.3, mime-types@2.1.3)
├┼┼ type-is@1.6.5 (media-typer@0.3.0, mime-types@2.1.3)
```

Having done that, you should now have a directory that contains all of the code required for this example.

NOTE The `npm install` step must be run separately for every exercise.

Each of the exercises contains three JavaScript source files: `client.js`, `authorizationServer.js`, and `protectedResource.js`, along with other support files and libraries to make things run. Each of these needs to be run separately using the `node` command, and we suggest running each in a different terminal window to avoid

[3] https://github.com/oauthinaction/oauth-in-action-code/
[4] https://www.manning.com/books/oauth-2-in-action

[redacted]

confusing the log files. The order of starting them up does not matter, but they do have to be running at the same time in order for most example programs to work.

For example, running the client application should produce output like the following in the terminal window:

```
> node client.js
OAuth Client is listening at http://127.0.0.1:9000
```

The authorization server is launched like this:

```
> node authorizationServer.js
OAuth Authorization Server is listening at http://127.0.0.1:9001
```

And the protected resource is launched like this:

```
> node protectedResource.js
OAuth Protected Resource is listening at http://127.0.0.1:9002
```

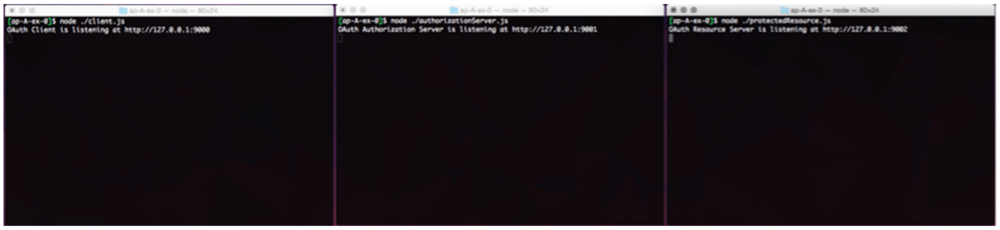
We recommend running all three in separate terminal windows so that you can see the output of the programs as they run. See figure A.1.

Each component is set up to run on a different port on `localhost`, in a separate process:

- The OAuth Client application (client.js) runs on `http://localhost:9000/`
- The OAuth Authorization Server application (authorizationServer.js) runs on `http://localhost:9001/`
- The OAuth Protected Resource Application (protectedResource.js) runs on `http://localhost:9002/`

All of the applications have been set up to serve static files such as images and Cascading Style Sheets (CSS). These are included in the `files` directory in the project and won't need to be edited for any of the exercises. In addition, there are HTML templates in the `files` directory. These are used in the applications to generate HTML pages based on variable inputs. When templates are used, they're set up at the beginning of the application with the following code:

```
app.engine('html', cons.underscore);
app.set('view engine', 'html');
app.set('views', 'files');
```



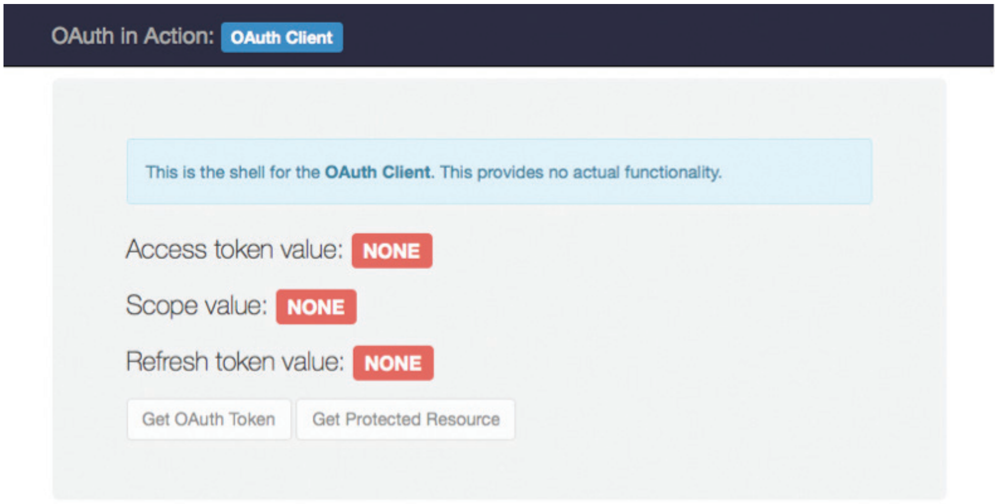
```
[ap-A-ex-0]$ node ./client.js
OAuth Client is listening at http://127.0.0.1:9000
```

```
[ap-A-ex-0]$ node ./authorizationServer.js
OAuth Authorization Server is listening at http://127.0.0.1:9001
```

```
[ap-A-ex-0]$ node ./protectedResource.js
OAuth Resource Server is listening at http://127.0.0.1:9002
```

Figure A.1 Three terminal windows running each component in parallel

[redacted]



This is the shell for the **OAuth Client**. This provides no actual functionality.

Access token value: NONE

Scope value: NONE

Refresh token value: NONE

Figure A.2 The homepage for the client shell

The templates won't need to be edited during the exercises, but they will occasionally be reviewed to showcase functionality. We're using the Underscore.js[5] templating system along with the Consolidate.js[6] library to create and manage all of the templates in all of the examples. You can pass variables to these templates and have them render their output using the render call on the response object, like this:

```
res.render('index', {access_token: access_token});
```

The three code files in this first example contain no actual functionality, but if you can see the welcome page for each then you'll know that the applications are running correctly and the dependencies have been installed. For example, visiting the OAuth Client's URL `http://localhost:9000/` in a web browser on your machine should produce what you see in figure A.2.

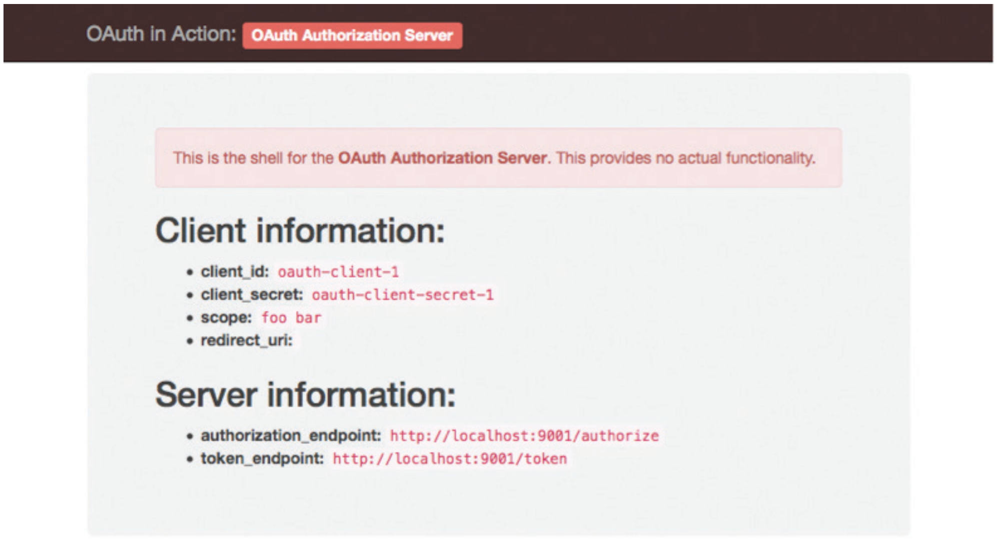
Likewise, the authorization server on `http://localhost:9001/` will look like what you see in figure A.3.

And finally, the protected resource on `http://localhost:9002/` will display what is shown in figure A.4 (note that the protected resource doesn't usually have a user-facing component).

To add HTTP handlers to our applications, we need to add them as *routes* to the Express.js application object. In each route, we tell the application which HTTP methods to listen for, which URL patterns to listen for, and which function to call when these conditions are matched. The function is passed a request object and a response

[5] http://underscorejs.org/
[6] https://github.com/tj/consolidate.js

[redacted]



This is the shell for the **OAuth Authorization Server**. This provides no actual functionality.

Client information:

- **client_id:** `oauth-client-1`
- **client_secret:** `oauth-client-secret-1`
- **scope:** `foo bar`
- **redirect_uri:**

Server information:

- **authorization_endpoint:** `http://localhost:9001/authorize`
- **token_endpoint:** `http://localhost:9001/token`

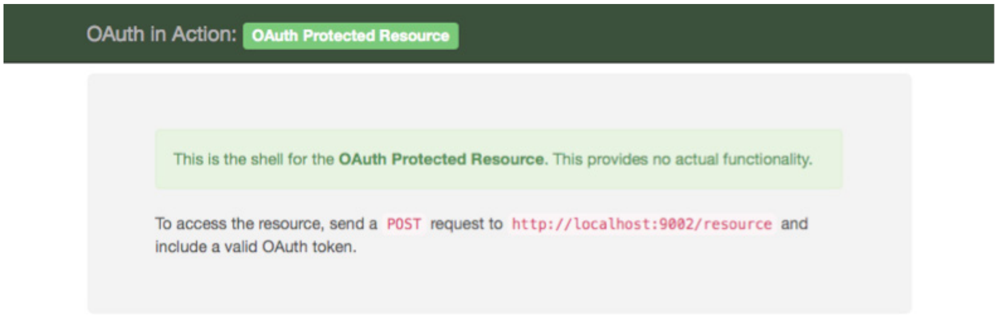
Figure A.3 The homepage for the authorization server shell

object as parameters. For instance, this example listens for HTTP `GET` requests on `/foo` and calls the given anonymous function.

```
app.get('/foo', function (req, res) {

});
```

We'll be following the convention of referring to the request object as `req` and the response object as `res` throughout our exercises. The request object contains information about the incoming HTTP request, including headers, URL, query parameters, and other things that correspond to the incoming request. The response object is used to send information back in the HTTP response, including a status code, headers, a response body, and more.



This is the shell for the **OAuth Protected Resource**. This provides no actual functionality.

To access the resource, send a `POST` request to `http://localhost:9002/resource` and include a valid OAuth token.

Figure A.4 The homepage for the protected resource shell

[redacted]

We'll be storing a lot of stateful information in global variables, declared at the top of the respective file. In any reasonable web application, all of these pieces of state would be tied to the user's session instead of global variables in the application itself. A native application is likely to use an approach similar to our framework and rely on the features of the host operating system to provide local user session authentication.

You'll use this simple framework throughout the book to build OAuth clients, protected resources, and authorization servers. For the most part, each exercise will come set up with things nearly complete, and you'll be required only to fill in the small bits of OAuth-related functionality that we're discussing in a given exercise.

For every exercise, you'll be able to find the completed code in the `completed` directory under the exercise directory. If you get stuck, we recommend opening up these files to take a look at how the "official" answer works.

[redacted]

appendix B *Extended code listings*

This appendix contains extended code listings for the exercises throughout the book. In the chapters themselves, we've attempted to focus on the parts of the code necessary for functionality. We don't replicate the whole code base, since that's available on GitHub for your perusal at any time. However, it's often useful to see the multiple snippets of code talked about in a chapter in a slightly larger context than is viable within a chapter. Here we list some of these larger functions referenced throughout the book.

Listing 1 Authorization request function (3-1)

```
app.get('/authorize', function(req, res){

   access_token = null;

   state = randomstring.generate();

   var authorizeUrl = buildUrl(authServer.authorizationEndpoint, {
         response_type: 'code',
         client_id: client.client_id,
         redirect_uri: client.redirect_uris[0],
         state: state
   });

   console.log("redirect", authorizeUrl);
   res.redirect(authorizeUrl);
});
```

[redacted]

Listing 2 Callback and token request (3-1)

```
app.get('/callback', function(req, res){

   if (req.query.error) {
        res.render('error', {error: req.query.error});
        return;
   }

   if (req.query.state != state) {
        console.log('State DOES NOT MATCH: expected %s got %s', state, req.
        query.state);
        res.render('error', {error: 'State value did not match'});
        return;
   }

   var code = req.query.code;

   var form_data = qs.stringify({
        grant_type: 'authorization_code',
        code: code,
        redirect_uri: client.redirect_uris[0]
   });
   var headers = {
        'Content-Type': 'application/x-www-form-urlencoded',
        'Authorization': 'Basic ' + encodeClientCredentials(client.client_id,
        client.client_secret)
   };

   var tokRes = request('POST', authServer.tokenEndpoint, {
                  body: form_data,
                  headers: headers
   });

   console.log('Requesting access token for code %s',code);

   if (tokRes.statusCode >= 200 && tokRes.statusCode < 300) {
        var body = JSON.parse(tokRes.getBody());

        access_token = body.access_token;
        console.log('Got access token: %s', access_token);

        res.render('index', {access_token: access_token, scope: scope});
   } else {
        res.render('error', {error: 'Unable to fetch access token, server
        response: ' + tokRes.statusCode})
   }
});
```

Listing 3 Fetching a protected resource (3-1)

```
app.get('/fetch_resource', function(req, res) {

   if (!access_token) {
        res.render('error', {error: 'Missing Access Token'});
```

[redacted]

```
  }

  console.log('Making request with access token %s', access_token);

  var headers = {
       'Authorization': 'Bearer ' + access_token
  };

  var resource = request('POST', protectedResource,
       {headers: headers}
  );

  if (resource.statusCode >= 200 && resource.statusCode < 300) {
       var body = JSON.parse(resource.getBody());
       res.render('data', {resource: body});
       return;
  } else {
       access_token = null;
       res.render('error', {error: resource.statusCode});
       return;
  }


});
```

Listing 4 Refreshing an access token (3-2)

```
app.get('/fetch_resource', function(req, res) {

  console.log('Making request with access token %s', access_token);

  var headers = {
       'Authorization': 'Bearer ' + access_token,
       'Content-Type': 'application/x-www-form-urlencoded'
  };

  var resource = request('POST', protectedResource,
       {headers: headers}
  );

  if (resource.statusCode >= 200 && resource.statusCode < 300) {
       var body = JSON.parse(resource.getBody());
       res.render('data', {resource: body});
       return;
  } else {
       access_token = null;
       if (refresh_token) {
               refreshAccessToken(req, res);
               return;
       } else {
               res.render('error', {error: resource.statusCode});
               return;
       }
  }


});
```

[redacted]

```
var refreshAccessToken = function(req, res) {
   var form_data = qs.stringify({
        grant_type: 'refresh_token',
        refresh_token: refresh_token
   });
   var headers = {
        'Content-Type': 'application/x-www-form-urlencoded',
        'Authorization': 'Basic ' + encodeClientCredentials(client.client_id,
        client.client_secret)
   };
   console.log('Refreshing token %s', refresh_token);
   var tokRes = request('POST', authServer.tokenEndpoint, {
                body: form_data,
                headers: headers
   });
   if (tokRes.statusCode >= 200 && tokRes.statusCode < 300) {
        var body = JSON.parse(tokRes.getBody());

        access_token = body.access_token;
        console.log('Got access token: %s', access_token);
        if (body.refresh_token) {
                refresh_token = body.refresh_token;
                console.log('Got refresh token: %s', refresh_token);
        }
        scope = body.scope;
        console.log('Got scope: %s', scope);

        res.redirect('/fetch_resource');
        return;
   } else {
        console.log('No refresh token, asking the user to get a new access
          token');
        refresh_token = null;
        res.render('error', {error: 'Unable to refresh token.'});
        return;
   }
};
```

Listing 5 Extracting the access token (4-1)

```
var getAccessToken = function(req, res, next) {

   var inToken = null;
   var auth = req.headers['authorization'];
   if (auth && auth.toLowerCase().indexOf('bearer') == 0) {
        inToken = auth.slice('bearer '.length);
   } else if (req.body && req.body.access_token) {
        inToken = req.body.access_token;
   } else if (req.query && req.query.access_token) {
        inToken = req.query.access_token
   }
};
```

[redacted]

Listing 6 Looking up the token (4-1)

```
var getAccessToken = function(req, res, next) {

   var inToken = null;
   var auth = req.headers['authorization'];
   if (auth && auth.toLowerCase().indexOf('bearer') == 0) {
        inToken = auth.slice('bearer '.length);
   } else if (req.body && req.body.access_token) {
        inToken = req.body.access_token;
   } else if (req.query && req.query.access_token) {
        inToken = req.query.access_token
   }

   console.log('Incoming token: %s', inToken);
   nosql.one(function(token) {
        if (token.access_token == inToken) {
               return token;
        }
   }, function(err, token) {
        if (token) {
               console.log("We found a matching token: %s", inToken);
        } else {
               console.log('No matching token was found.');
        }
        req.access_token = token;
        next();
        return;
   });
};
```

Listing 7 Authorization endpoint (5-1)

```
app.get("/authorize", function(req, res){

   var client = getClient(req.query.client_id);

   if (!client) {
        console.log('Unknown client %s', req.query.client_id);
        res.render('error', {error: 'Unknown client'});
        return;
   } else if (!__.contains(client.redirect_uris, req.query.redirect_uri))
   {
        console.log('Mismatched redirect URI, expected %s got %s',
        client.redirect_uris, req.query.redirect_uri);
        res.render('error', {error: 'Invalid redirect URI'});
        return;
   } else {

        var reqid = randomstring.generate(8);

        requests[reqid] = req.query;

        res.render('approve', {client: client, reqid: reqid });
        return;
   }

});
```

[redacted]

Listing 8 Handling user approval (5-1)

```
app.post('/approve', function(req, res) {

  var reqid = req.body.reqid;
  var query = requests[reqid];
  delete requests[reqid];

  if (!query) {
       res.render('error', {error: 'No matching authorization request'});
       return;
  }

  if (req.body.approve) {
       if (query.response_type == 'code') {
               var code = randomstring.generate(8);

               codes[code] = { request: query };

               var urlParsed = buildUrl(query.redirect_uri, {
                      code: code,
                      state: query.state
               });
               res.redirect(urlParsed);
               return;
       } else {
               var urlParsed = buildUrl(query.redirect_uri, {
                      error: 'unsupported_response_type'
               });
               res.redirect(urlParsed);
               return;
       }
  } else {
       var urlParsed = buildUrl(query.redirect_uri, {
               error: 'access_denied'
       });
       res.redirect(urlParsed);
       return;
  }

});
```

Listing 9 Token endpoint (5-1)

```
app.post("/token", function(req, res){

  var auth = req.headers['authorization'];
  if (auth) {
       var clientCredentials = decodeClientCredentials(auth);
       var clientId = clientCredentials.id;
       var clientSecret = clientCredentials.secret;
  }

  if (req.body.client_id) {
       if (clientId) {
               console.log('Client attempted to authenticate with multiple
               methods');
```

[redacted]

```
            res.status(401).json({error: 'invalid_client'});
            return;
        }

        var clientId = req.body.client_id;
        var clientSecret = req.body.client_secret;
    }

    var client = getClient(clientId);
    if (!client) {
        console.log('Unknown client %s', clientId);
        res.status(401).json({error: 'invalid_client'});
        return;
    }

    if (client.client_secret != clientSecret) {
        console.log('Mismatched client secret, expected %s got %s',
        client.client_secret, clientSecret);
        res.status(401).json({error: 'invalid_client'});
        return;
    }

    if (req.body.grant_type == 'authorization_code') {

        var code = codes[req.body.code];

        if (code) {
            delete codes[req.body.code]; // burn our code, it's been used
            if (code.request.client_id == clientId) {

                var access_token = randomstring.generate();
                nosql.insert({ access_token: access_token, client_id:
                clientId });

                console.log('Issuing access token %s', access_token);

                var token_response = { access_token: access_token,
                token_type: 'Bearer' };

                res.status(200).json(token_response);
                console.log('Issued tokens for code %s', req.body.
                code);

                return;
            } else {
                console.log('Client mismatch, expected %s got %s',
                code.request.client_id, clientId);
                res.status(400).json({error: 'invalid_grant'});
                return;
            }
        } else {
            console.log('Unknown code, %s', req.body.code);
            res.status(400).json({error: 'invalid_grant'});
            return;
        }
```

[redacted]

```
  } else {
      console.log('Unknown grant type %s', req.body.grant_type);
      res.status(400).json({error: 'unsupported_grant_type'});
  }
});
```

Listing 10 Refreshing access tokens (5-2)

```
} else if (req.body.grant_type == 'refresh_token') {
  nosql.one(function(token) {
      if (token.refresh_token == req.body.refresh_token) {
          return token;
      }
  }, function(err, token) {
      if (token) {
          console.log("We found a matching refresh token: %s", req.body.
          refresh_token);
          if (token.client_id != clientId) {
              nosql.remove(function(found) { return (found == token);
              }, function () {} );
              res.status(400).json({error: 'invalid_grant'});
              return;
          }
          var access_token = randomstring.generate();
          nosql.insert({ access_token: access_token, client_id:
          clientId });
          var token_response = { access_token: access_token, token_type:
          'Bearer',  refresh_token: token.refresh_token };
          res.status(200).json(token_response);
          return;
      } else {
          console.log('No matching token was found.');
          res.status(400).json({error: 'invalid_grant'});
          return;
      }
  });
```

Listing 11 Introspection endpoint (11-3)

```
app.post('/introspect', function(req, res) {
  var auth = req.headers['authorization'];
  var resourceCredentials = decodeClientCredentials(auth);
  var resourceId = resourceCredentials.id;
  var resourceSecret = resourceCredentials.secret;

  var resource = getProtectedResource(resourceId);
  if (!resource) {
      console.log('Unknown resource %s', resourceId);
      res.status(401).end();
      return;
  }

  if (resource.resource_secret != resourceSecret) {
```

[redacted]

```
       console.log('Mismatched secret, expected %s got %s', resource.
       resource_secret, resourceSecret);
       res.status(401).end();
       return;
  }

  var inToken = req.body.token;
  console.log('Introspecting token %s', inToken);
  nosql.one(function(token) {
       if (token.access_token == inToken) {
               return token;
       }
  }, function(err, token) {
       if (token) {
               console.log("We found a matching token: %s", inToken);

               var introspectionResponse = {
                      active: true,
                      iss: 'http://localhost:9001/',
                      aud: 'http://localhost:9002/',
                      sub: token.user ? token.user.sub : undefined,
                      username: token.user ? token.user.preferred_username :
                      undefined,
                      scope: token.scope ? token.scope.join(' ') : undefined,
                      client_id: token.client_id
               };

               res.status(200).json(introspectionResponse);
               return;
       } else {
               console.log('No matching token was found.');

               var introspectionResponse = {
                      active: false
               };
               res.status(200).json(introspectionResponse);
               return;
       }
  });


});
```

Listing 12 Token revocation endpoint (11-5)

```
app.post('/revoke', function(req, res) {
  var auth = req.headers['authorization'];
  if (auth) {
       // check the auth header
       var clientCredentials = decodeClientCredentials(auth);
       var clientId = clientCredentials.id;
       var clientSecret = clientCredentials.secret;
  }

  // otherwise, check the post body
```

[redacted]

```
  if (req.body.client_id) {
       if (clientId) {
               // if we've already seen the client's credentials in the
               authorization header, this is an error
               console.log('Client attempted to authenticate with multiple
               methods');
               res.status(401).json({error: 'invalid_client'});
               return;
       }

       var clientId = req.body.client_id;
       var clientSecret = req.body.client_secret;
  }

  var client = getClient(clientId);
  if (!client) {
       console.log('Unknown client %s', clientId);
       res.status(401).json({error: 'invalid_client'});
       return;
  }

  if (client.client_secret != clientSecret) {
       console.log('Mismatched client secret, expected %s got %s', client.
       client_secret, clientSecret);
       res.status(401).json({error: 'invalid_client'});
       return;
  }

  var inToken = req.body.token;
  nosql.remove(function(token) {
       if (token.access_token == inToken && token.client_id == clientId) {
               return true;
       }
  }, function(err, count) {
       console.log("Removed %s tokens", count);
       res.status(204).end();
       return;
  });

});
```

Listing 13 Registration endpoint (12-1)

```
app.post('/register', function (req, res){

  var reg = {};

  if (!req.body.token_endpoint_auth_method) {
       reg.token_endpoint_auth_method = 'secret_basic';
  } else {
       reg.token_endpoint_auth_method = req.body.token_endpoint_auth_method;
  }

  if (!__.contains(['secret_basic', 'secret_post', 'none'], reg.token_
  endpoint_auth_method)) {
```

[redacted]

```
     res.status(400).json({error: 'invalid_client_metadata'});
     return;
}

if (!req.body.grant_types) {
     if (!req.body.response_types) {
            reg.grant_types = ['authorization_code'];
            reg.response_types = ['code'];
     } else {
            reg.response_types = req.body.response_types;
            if (__.contains(req.body.response_types, 'code')) {
                   reg.grant_types = ['authorization_code'];
            } else {
                   reg.grant_types = [];
            }
     }
} else {
     if (!req.body.response_types) {
            reg.grant_types = req.body.grant_types;
            if (__.contains(req.body.grant_types, 'authorization_code')) {
                   reg.response_types =['code'];
            } else {
                   reg.response_types = [];
            }
     } else {
            reg.grant_types = req.body.grant_types;
            reg.reponse_types = req.body.response_types;
            if (__.contains(req.body.grant_types, 'authorization_code') &&
            !__.contains(req.body.response_types, 'code')) {
                   reg.response_types.push('code');
            }
            if (!__.contains(req.body.grant_types, 'authorization_code')
            && __.contains(req.body.response_types, 'code')) {
                   reg.grant_types.push('authorization_code');
            }
     }
}

if (!__.isEmpty(__.without(reg.grant_types, 'authorization_code',
'refresh_token')) ||
     !__.isEmpty(__.without(reg.response_types, 'code'))) {
     res.status(400).json({error: 'invalid_client_metadata'});
     return;
}

if (!req.body.redirect_uris || !__.isArray(req.body.redirect_uris) ||
__.isEmpty(req.body.redirect_uris)) {
     res.status(400).json({error: 'invalid_redirect_uri'});
     return;
} else {
     reg.redirect_uris = req.body.redirect_uris;
}

if (typeof(req.body.client_name) == 'string') {
     reg.client_name = req.body.client_name;
}
```

[redacted]

```
 if (typeof(req.body.client_uri) == 'string') {
    reg.client_uri = req.body.client_uri;
 }

 if (typeof(req.body.logo_uri) == 'string') {
    reg.logo_uri = req.body.logo_uri;
 }

 if (typeof(req.body.scope) == 'string') {
    reg.scope = req.body.scope;
 }

 reg.client_id = randomstring.generate();
 if (__.contains(['client_secret_basic', 'client_secret_post']), reg.token_
 endpoint_auth_method) {
    reg.client_secret = randomstring.generate();
 }

 reg.client_id_created_at = Math.floor(Date.now() / 1000);
 reg.client_secret_expires_at = 0;

 clients.push(reg);

 res.status(201).json(reg);
 return;
});
```

Listing 14 UserInfo endpoint (13-1)

```
var userInfoEndpoint = function(req, res) {

 if (!__.contains(req.access_token.scope, 'openid')) {
    res.status(403).end();
    return;
 }

 var user = req.access_token.user;
 if (!user) {
    res.status(404).end();
    return;
 }

 var out = {};
 __.each(req.access_token.scope, function (scope) {
    if (scope == 'openid') {
        __.each(['sub'], function(claim) {
            if (user[claim]) {
                out[claim] = user[claim];
            }
        });
    } else if (scope == 'profile') {
        __.each(['name', 'family_name', 'given_name', 'middle_name',
        'nickname', 'preferred_username', 'profile', 'picture',
        'website', 'gender', 'birthdate', 'zoneinfo', 'locale',
        'updated_at'], function(claim) {
            if (user[claim]) {
```

[redacted]

```
                              out[claim] = user[claim];
                       }
               });
       } else if (scope == 'email') {
               __.each(['email', 'email_verified'], function(claim) {
                       if (user[claim]) {
                              out[claim] = user[claim];
                       }
               });
       } else if (scope == 'address') {
               __.each(['address'], function(claim) {
                       if (user[claim]) {
                              out[claim] = user[claim];
                       }
               });
       } else if (scope == 'phone') {
               __.each(['phone_number', 'phone_number_verified'],
               function(claim) {
                       if (user[claim]) {
                              out[claim] = user[claim];
                       }
               });
       }
  });

  res.status(200).json(out);
  return;
};
```

Listing 15 Processing the ID token (13-1)

```
if (body.id_token) {
  userInfo = null;
  id_token = null;

  console.log('Got ID token: %s', body.id_token);

  var pubKey = jose.KEYUTIL.getKey(rsaKey);
  var tokenParts = body.id_token.split('.');
  var payload = JSON.parse(base64url.decode(tokenParts[1]));
  console.log('Payload', payload);
  if (jose.jws.JWS.verify(body.id_token, pubKey, [rsaKey.alg])) {
       console.log('Signature validated.');
       if (payload.iss == 'http://localhost:9001/') {
               console.log('issuer OK');
               if ((Array.isArray(payload.aud) && __.contains(payload.aud,
               client.client_id)) ||
                       payload.aud == client.client_id) {
                       console.log('Audience OK');

                       var now = Math.floor(Date.now() / 1000);

                       if (payload.iat <= now) {
                              console.log('issued-at OK');
                              if (payload.exp >= now) {
                                      console.log('expiration OK');
```

[redacted]

```
                                     console.log('Token valid!');

                                     id_token = payload;

                              }
                       }
               }
        }
  }
  res.render('userinfo', {userInfo: userInfo, id_token: id_token});
  return;
}
```

Listing 16 Introspecting and verifying a PoP token (15-1)

```
var getAccessToken = function(req, res, next) {
  var auth = req.headers['authorization'];
  var inToken = null;
  if (auth && auth.toLowerCase().indexOf('pop') == 0) {
       inToken = auth.slice('pop '.length);
  } else if (req.body && req.body.pop_access_token) {
       inToken = req.body.pop_access_token;
  } else if (req.query && req.query.pop_access_token) {
       inToken = req.query.pop_access_token
  }

  console.log('Incoming PoP: %s', inToken);
  var tokenParts = inToken.split('.');
  var header = JSON.parse(base64url.decode(tokenParts[0]));
  var payload = JSON.parse(base64url.decode(tokenParts[1]));

  console.log('Payload', payload);

  var at = payload.at;
  console.log('Incmoing access token: %s', at);

  var form_data = qs.stringify({
       token: at
  });
  var headers = {
       'Content-Type': 'application/x-www-form-urlencoded',
       'Authorization': 'Basic ' +
       encodeClientCredentials(protectedResource.resource_id,
       protectedResource.resource_secret)
  };

  var tokRes = request('POST', authServer.introspectionEndpoint, {
       body: form_data,
       headers: headers
  });

  if (tokRes.statusCode >= 200 && tokRes.statusCode < 300) {
       var body = JSON.parse(tokRes.getBody());

       console.log('Got introspection response', body);
       var active = body.active;
       if (active) {
```

[redacted]

```
               var pubKey = jose.KEYUTIL.getKey(body.access_token_key);
               if (jose.jws.JWS.verify(inToken, pubKey, [header.alg])) {
                       console.log('Signature is valid');

                       if (!payload.m || payload.m == req.method) {
                               if (!payload.u || payload.u ==
                                     'localhost:9002') {
                                      if (!payload.p || payload.p == req.path)
                                             {
                                              console.log('All components
                                              matched');


                                              req.access_token = {
                                                      access_token: at,
                                                      scope: body.scope
                                              };

                                      }
                               }
                       }

               }

       }
  }
  next();
  return;

};
```

[redacted]

[redacted]